Maintain-e-nator: *Make the World a Better Place*

Xin Liu
University at Buffalo
1932 Wallamaloo Lane
xliu36@buffalo.edu

John Longanecker
University at Buffalo
P.O. Box 1212
john.longanecker@gmail.com

Juehui Zhang
University at Buffalo
Hekla, Iceland
larst@affiliation.org

ABSTRACT

Our system seeks to improve the quality of a facility as well as decrease an organization's overall operating expenses. By letting those who maintain facilities know about problems sooner. Maintenance workers can react quicker and more efficiently if they have better information about the status of their facilities. *Maintain-e-nator* provides a cell phone application to report problems as well as a web interface to allow workers to manage and be notified of problems.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Applications

1. INTRODUCTION

Eventually everything breaks. Nothing lasts forever. Buildings start as brand new but eventually break down. Roads start as smooth, but eventually develop potholes. These breakdowns can sometimes be ignored like a squeaky door, but others can cause safety and health risks. If the stairs of a building are in disrepair they could cause a tripping hazard for other people. The reasons for something not being fixed can vary. Budget, time, priority and awareness all play a role as to when thing get repaired. We seek to streamline the process at which maintainers become aware of problems.

2. MOTIVATIONS

It is easy for a large organization to be unaware of all the maintenance problems that their facilities have. Some obscure room may need a light bulb replaced but those that work in that area do not report the problem or are not around when the problem is exhibiting its behaviors. So the people who take a night class are the only ones aware of the problem. They do not know where to submit a problem and are often not willing to do the necessary research to find out how to report a problem.

These problems are not limited to the inside of buildings. They can also involve roads, landscaping, sidewalks, and outdoor sports facilities. For example a pallet could fall off the back of a truck and be sitting in the middle of a road. Most drivers will drive around the pallet and not report it. People are not sure how to report a problem like this or it is not convenient to notify the people who maintain the road. Wouldn't it be nice if one could take out their cellphone and submit a report? Our version of *Maintain-e-Nator* is targeted for the University at Buffalo's North Campus, but can easily be configured to be used in any setting.

2.1 What is a problem?

So far we have talked about what a problem is, but have yet to define what constitutes a problem. We consider a maintenance problem to be anything that can hurt someone as well as anything that detracts from the overall quality of a facility. At the end of the day the users who submit a problem are the ones who are deciding what a problem is. Who better to determine a problem then those who actually use the facilities?

2.2 Goal

The overall goal is to make maintenance workers aware of the problems that exists on their property. Are android application and web interface will not promise cleaner facilities. Our goal is to help those that manage a property.

3. DESIGN

Our application comprises of two parts: the Android application and the backend web application, which are designed for two kinds of users: `submitters` and `maintainers`. Data flow can be depicted as Fig. ....

The data flow of our application is fairly intuitive: whenever a submitter spots some issues on a facility, they open up the Android application, which will try to locate their position and generate a meaningful location (such as a building name or the nearest mailing address), the submitters can take a few photos of the issue and record some audio regarding to the issue, then just hit the submit button, that's it! The rest for maintainers, they can use the web application to organize all the submitted issues, log their progresses on repair the issues. And if submitters choose to report the issues with their Google accounts, the backend can also update the

issues status to submitters via email notifications automatically.

3.1 Android Application

We developed our client application on the Android [**?**] platform, it is mainly consisted of three modules: 1. login module 2. positioning module 3. information collecting module. Below we talk about these three modules in detail.

3.1.1 Login

Whenever some submitter wants to submit issues and open the application, she can choose to log in the application with her Google account or anonymously. We choose Google account because it is available in (or necessary for) every Android device. We add some personal settings for users logs in with their Google accounts, and once the user submit some issue, the application will also pass her personal information (just name and email address) to our web application, which maybe used in the future for contacting with the user. With the `AccountManager` in Android library, we can only get the submitter's email address. To make our app more personalize, we want a little more of submitter's profile information. Once the submitter grant the application permission to retrieve her profile with the access token, we can use it to request the submitter information via calling UserInfo Google API [**?**]. This whole process follows the `OAuth2` specification [**?**]. The application doesn't try to remember how the submitter logged in last time and use it for next time, each time submitter can choose whatever manner she wants.

3.1.2 Positioning

Each time when a submitter opens the application and files an issue, the application will start to request a single location update via Network or GPS. We prefer Network update over GPS for it is much faster and accurate enough with the omni Wi-Fi APs within our campus. Also since it is unlikely for the submitter to move around when submitting an issue, we only request *single* location update to save battery life. ...

Once the application gets the geolocation data, which is a pair of number for latitude and longitude, we need to convert this geolocation into some meaningful information for the submitter. if the submitter is indoor (actually we will get the textual locations for both indoor and outdoor scenarios), the application will try to guess which hall she maybe in by calculating the Euclidean distance between current geolocation and the predefined geolocations of halls we current support, and assume the one with the nearest distance is the hall the submitter is in. Otherwise, if the submitter is out of door, the application will try to get the meaningful location of the submitter by using Google Place API [**?**] with the geolocation data, Currently we just use the first result returned by this API.

Submitters can modify the location information suggested by the application later. Also, for outdoor scenario, we implemented a `MapView` with a draggable marker which indicates submitter current position. In some cases, if the geolocation provided is not that accurate or submitters move to another place after the location update, they can adjust the position of marker, then the application will use that new geolocation accordingly.

3.1.3 Information collecting

For future maintainers to locate the issue position easily and accurately, we hope submitters can report the issue information as detail as possible. Since the location detail for indoor and outdoor could be very different, to ease the process of filling the detail of an issue, the application has two different forms for indoor and outdoor separately, as shown in Fig **??**, The application requires submitter to at least provide a description, the location (which maybe acquired automatically as described in Sec. **??** or replaced by submitter). For current indoor experiment, we only support four halls - Davis Hall, Jarvis Hall, Furnas Hall and Ketter Hall.

Only textual information may not be detail enough when describing an issue, thus we encourage submitter to add photos and also an audio recording regarding the location or issue detail. By long-pressing the image area, the submitter can add up to three photos either from camera capture or gallery. After that the submitter can view the full-size photo by clicking it and decide whether to keep it or replace it with a new one. For the audio recording, to make it act like a two-way radio, we implement it as when submitter `holding` the recording button, the app will start record the audio until submitter release it. We choose `wav` format for saving the audio recording because it is the quick and dirty way to enable the recorded audio be played back both on the Android device and the browser. The supported audio recording formats on Android (to our best knowledge), e.g., `mp4, 3gp` are not supported by current browsers. While the other formats supported by browsers without any plugins, e.g., `mp3, ogg` need third-party native libraries to be recorded on Android devices. For `wav` format, after we collecting the raw audio data, we just need to wrap it up with `wav` header, while the tradeoff is that file size is larger compared to those compressed formats.

3.1.4 Others

The application will save all the issues reported by the submitter locally on the Android device if the submitter choose to. Submitter can later view the details of all the issues she reported before. We also add some animations to activity switching, users' interaction to make the application more accessible.

3.2 Web Application

For backend web service[1], we host our web application on `heroku` [2], which is a cloud application platform that saves us from all kinds of pitfalls in servers management, deployment. Since `heroku` does not store static files as well as users upload files, we use `Amazon S3` [3] for this task.

We use `Django` [**?**] web framework for our web application, its active community makes it easy and fast to develop web apps with numerous open source modules (a full list of all open source modules we used [4]). Our web application mainly has two components, private admin site, which is for

[1] `http://maintain-e-nator.herokuapp.com`
[2] `https://www.heroku.com/`
[3] `https://aws.amazon.com/s3/`
[4] `https://github.com/forkloop/Maintainenator-backend/blob/master/requirements.txt`

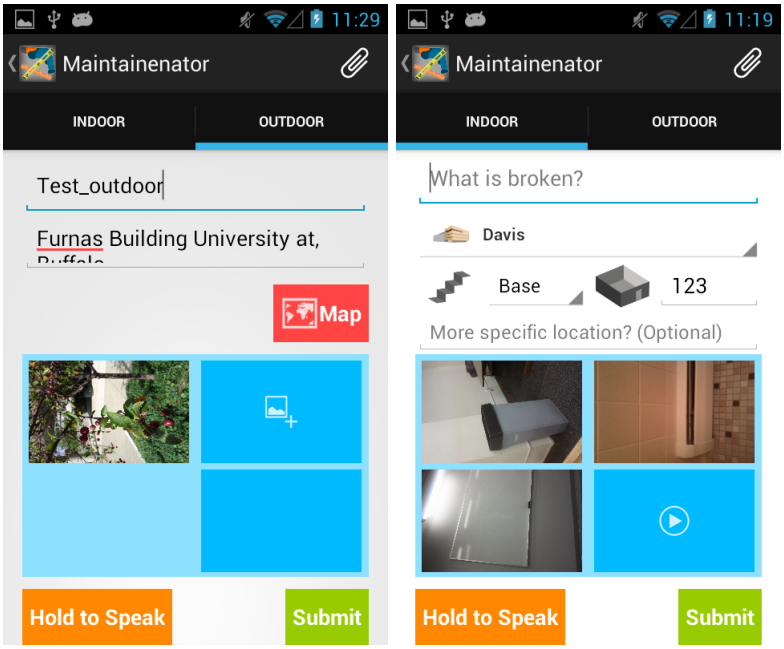

Figure 1: Issue form.

maintainers to manage all the reported issues, and public site, which is for every one to view existing reported issues.

3.2.1 *Private admin site*

`Django` is already packed with an admin module, which however is not that fancy, We instead use `grappelli` [5] for polishing up the admin interface. The admin site is the main entry for maintainers, which allows multiple maintainers to manage the reported issues, filter them based on the issues location, severity level, status, submitted date *etc.* With the embedded Google Maps, maintainers can get an better idea of where the issues are for outdoor problems. Also, the submitted photos and audio recording can give them more direct intuitions of what the issues are. Maintainers can also keep tracking of their working progress for issues by add logs. Once there is an update for an issue, the web application will send an email notification to the submitter if there is one. Currently, we only send notification to submitter once the issue is fixed.

3.2.2 *Public site*

The web application also has a public site that is intended for all to browse existing issues. Inspired by `Pinterest, Airbnb` and other similar websites, we work on this public site to make it as an Single-page application [**?**], which can offer users a better experience with the infinite-scrolling, no more page reloading when viewing different issues. This is done by adding a model-view-controller (MVC) layer to user browser, and we use `Backbone.js` [6] as our frontend MVC. When user first open this public site, it starts to render a page and fetch the required data asynchronously from web application RESTful API, with user scrolling down for more, it continues pre-fetching more data. Its ability to redraw any part of the UI without requiring a server roundtrip to retrieve HTML makes it more-native-app-like [**?**]. The whole interaction with the backend is via the web application RESTful JSON interface.

3.2.3 *Others*

The web application provides a RESTful API, which the Android application uses to submit an issue, or the public site uses to retrieve the issues information. We use `tastypie` [7] to build this REST-style interface. We did little hack on `tastypie` for this API to support file upload via `mixin`.

Because `heroku` doesn't store user uploaded files, we use `Amazon S3` to store the photos and audio recording uploaded by submitters. With `boot`[8], we can achieve the transfer of uploaded files from `Heroku` to `Amazon S3` transparently, once the web application receives a file from submitter, it is written to `Amazon S3` directly.

Normally, for a web application, besides handling income requests, it has other tasks like preprocessing uploaded images, sending emails to registered users, and you don't want to mix these kinds of slow processes with HTTP handlers, for this would greatly slow down the request-response cycle, jeopardize users experience. Thus we separate these kinds of tasks to other processes, and use `foreman`[9] to manage them all. There are two kinds of processes for our web application, the `web process` is for responding all HTTP requests, either rendering a view or return some JSON data. While the `worker process` is for all other not emergent tasks, *e.g.*, sending submitters emails regarding the issues they submit-

[5] `http://www.grappelliproject.com/`
[6] `http://backbonejs.org/`
[7] `http://tastypieapi.org/`
[8] `http://docs.pythonboto.org/`
[9] `http://ddollar.github.com/foreman/`

ted.

4. TEST

We developed and tested our Android application on Google Nexus S with 4.1.2 version. The Android application should also work fine on the 4.0.* platform. The location suggested [...]The source codes for both Android application[10] and web application[11] are available on `github`.

5. FUTURE WORK

Though our application ..., there are a lot room for future improvement. 1. Current we didn't compress the photos and audio recording when sending them to the web application, which maybe consume a lot of data if the phone doesn't connect to Wi-Fi. To fix this, we can resize down the photos, and use a compressed audio format for recording the audio. Or we can allow submitters save current report and submit it later when they have Wi-Fi connection. 2. The web application admin site is still ..., some functions like collapse the same issues reported by different submitters, allow issues fixing assignment to maintainers. 3. We only tested the application within three of us, in the future, we can utilize `PhoneLab` [12] to fully test our application.

6. CONCLUSIONS

Maintenance issues have been an annoying problem for many years. People may find some issues nearby but they don't know how to report them. We propose a new android application called `Maintain-e-Nator`, to make it easy for people to report their issues. People can choose indoor or outdoor, write down the description of problem, take at most three pictures, and even do some recording. We also build a web application, which is used by maintainers to manage the reported issues, filtering them based on the location, severity level, status, submitted date etc. Maintainers can view the issues locations on Google Maps, get an idea of what the issues are via submitted photos and audio recording.

7. ACKNOWLEDGMENTS

We would like to thank Prof. Steven Y. Ko and Prof. Geoffrey Challen for their valuable suggestions. We would also like to thank PhoneLab for providing us Android phones.

[10] `https://github.com/forkloop/Maintain-e-nator`
[11] `https://github.com/forkloop/Maintainenator-backend`
[12] `http://www.phone-lab.org/`

APPENDIX

A. ACCOMPLISHMENTS

Here we list our accomplishments for each milestone, the slides for each milestone demo can been viewed in `speakerdeck`[13].

A.1 Milestone 1

- Set up the web application on `heroku`, has a simple public site.
- Develop the first version of Android application which is able to send some textual information and one photo to backend web application.

A.2 Milestone 2

- Rewrite `FormActivity` with `Fragment`, separate indoor and outdoor information form.
- Tweak on Android application, handle special scenarios when there is no location update, or network connection.
- Separate photo from issue model, allows multiple photos upload.
- Allow submitter to save submitted issues.
- Add admin site to web application.

A.3 Milestone 3

- Fully UI polishing for Android application, with designed icons, image buttons, new login activity, etc.
- Add audio recording function, allow submitter to also submit a short audio.
- Add a `MapView` to outdoor for submitter to adjust their location with Google Map interactively.
- Rewrite public site as Single-page application.

[13] `https://speakerdeck.com/forkloop/cse622-milestone-reports`